

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Cosimo et al.	§	Group Art Unit: 2626
	§	
Serial No. 10/721,435	§	Examiner: Neway, Samuel G.
	§	
Filed: November 25, 2003	§	Customer No.: 50170
	§	
For: Editor with Commands for	§	
Automatically Disabling and Enabling	§	
Program Code Portions	§	

**Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450**

ATTENTION: Board of Patent Appeals and Interferences

APPELLANTS' BRIEF (37 C.F.R. § 41.37)

This Appeal Brief is in furtherance of the Notice of Appeal filed March 20, 2007 (37 C.F.R. § 41.31).

The fees required under § 41.20(b)(2), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying Transmittal of Appeal Brief.

I. Real Party in Interest

The real party in interest in this appeal is the following party: International Business Machines Corporation.

II. Related Appeals and Interferences

With respect to other appeals and interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

III. Status of Claims

The status of the claims involved in this proceeding is as follows:

1. Claims canceled: 7
2. Claims withdrawing from consideration but not canceled: NONE
3. Claims pending: 1-6 and 8-21
4. Claims allowed: NONE
5. Claims rejected: 1-6 and 8-21

The claims on appeal are: claims 1-6 and 8-21.

IV. Status of Amendments

No amendments to the application were filed subsequent to mailing of the Final Office Action.

V. Summary of Claimed Subject Matter

The present invention, as recited in claim 1, provides a method of editing program code on a data processing system (e.g., 100 in Figure 1), the program code being suitable for subsequent processing. The method may comprise defining at least two portions of the program code, selecting a first defined portion of the at least two portions of the program code (e.g., Figure 3, functions (Func) and comments (Comm), see also page 12, lines 7-30), and compressing a representation of the first defined portion in a visual representation of the program code (e.g., page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The method may further comprise automatically disabling the first defined portion (e.g., block 340 in Figure 3; page 9, lines 4-14), the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

With regard to independent claim 9, the present invention provides a computer readable medium (e.g., see page 4, lines 14-19), having a computer readable program encoded thereon and being directly loadable into a working memory (e.g., see Figure 2) of a data processing system (e.g., 100 in Figure 1), the computer readable program having instructions for performing a method of editing program code (e.g., 205 in Figure 2) when the computer readable program is run on the data processing system (e.g., 100 in Figure 1), the program code being suitable for subsequent processing. The computer readable program may cause the data processing device to define at least two portions of the program code (e.g., Figure 3, functions (Func) and comments (Comm), see also page 12, lines 7-30), select a first defined portion of the at least two portions of the program code, and compress a representation of the first defined portion in a visual representation of the program code (e.g., page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains

visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The computer readable program may further cause the data processing system to automatically disable the first defined portion (e.g., block 340 in Figure 3; page 9, lines 4-14), the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

With regard to independent claim 10, the present invention provides a program product comprising a computer readable medium (e.g., see page 4, lines 14-19) on which a computer program is stored, the program being directly loadable into a working memory (e.g., see Figure 2) of a data processing system (e.g., 100 in Figure 1) for performing a method of editing program code (e.g., 205 in Figure 2) when the program is run on the data processing system (e.g., 100 in Figure 1), the program code being suitable for subsequent processing. The method, performed by the computer program may comprise defining at least two portions of the program code (e.g., Figure 3, functions (Func) and comments (Comm), see also page 12, lines 7-30), selecting a first defined portion of the at least two portions of the program code, and compressing a representation of the first defined portion in a visual representation of the program code (e.g., page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The method may further comprise automatically disabling the first defined portion (e.g., block 340 in Figure 3; page 9, lines 4-14), the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

Regarding independent claim 11, the present invention provides an editor (e.g., 205 in Figure 2) for editing program code on a data processing system (e.g., 100 in Figure 1), the program code being suitable for subsequent processing, wherein the editor (e.g., 205 in Figure 2) is provided as a computer readable program on a computer readable medium (e.g., see page 4, lines 14-19). The computer readable program may include software instructions for defining at least two portions of the program code (e.g., Figure 3, functions (Func) and comments (Comm),

see also page 12, lines 7-30), selecting a first defined portion of the at least two portions of the program code, and compressing a representation of the first defined portion in a visual representation of the program code (e.g., page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The software instructions may further automatically disable the first defined portion (e.g., block 340 in Figure 3; page 9, lines 4-14), the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

Regarding independent claim 12, the present invention provides a data processing system (e.g., 100 in Figure 1) for editing program code, the program code being suitable for subsequent processing. The system may include means for defining at least two portions of the program code (e.g., 220 and 205 in Figure 2; Figure 3, functions (Func) and comments (Comm), see also page 12, lines 7-30), means for selecting a first defined portion of the at least two portions of the program code (e.g., 220 and 205 in Figure 2), and means for compressing (e.g., 230 in Figure 2) a representation of the first defined portion in a visual representation of the program code (e.g., 225 in Figure 2; page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The system may further comprise means for automatically disabling (e.g., 230 in Figure 2; block 340 in Figure 3; page 9, lines 4-14) the first defined portion, the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

With regard to independent claim 13, the present invention provides a data processing system (e.g., 100 in Figure 1) for editing program code, the program code being suitable for subsequent processing. The system may comprise a processor, a memory coupled to the processor (e.g., 105 in Figure 1; page 4, lines 5-7), and an input device (e.g., 120 and/or 125 in

Figure 1). The memory may comprise instructions which, when executed by the processor, cause the processor to implement a software module for defining at least two portions of the program code (e.g., 220 and 205 in Figure 2; Figure 3, functions (Func) and comments (Comm), see also page 12, lines 7-30), a software module for selecting a first defined portion of the at least two portions of the program code in response to an input from the input device (e.g., 220 and 205 in Figure 2), and a software module for compressing (e.g., 230 in Figure 2) a representation of the first defined portion in a visual representation of the program code (e.g., 225 in Figure 2; page 12, lines 3-6) such that content of the first defined portion is not visible in the visual representation of the program code (e.g., compressor 230 in Figure 2; block 339 in Figure 3; page 8, line 33 to page 9, line 3; block 345 in Figure 3; page 9, lines 19-22). A second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code (e.g., block 357, 367, and/or 370 in Figure 3; page 9, lines 30-35). The instructions may further cause the processor to implement a software module for automatically disabling (e.g., 230 in Figure 2; block 340 in Figure 3; page 9, lines 4-14) the first defined portion, the disabled first defined portion being excluded from the subsequent processing (e.g., page 9, lines 10-12). The second defined portion is subjected to subsequent processing.

VI. Grounds of Rejection to be Reviewed on Appeal

The grounds of rejection to be reviewed on appeal are as follows:

- (1) Claims 2-3 and 5 stand rejected under 35 U.S.C. § 112, second paragraph as allegedly having insufficient antecedent basis for the feature “the at least one previously disabled portion”;
- (2) Claims 1, 6, 9-14, and 16-21 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Chenier (U.S. Patent Application Publication No. 2004/0003383) in view of Storisteneau (CA 2256931); and
- (3) Claims 2-5, 8, and 15 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Chenier in view of Storisteneau, and further in view of Endejan (U.S. Patent Application Publication No. 2002/0184611).

VII. Argument

A. Rejection under 35 U.S.C. § 112

The Final Office Action rejects claims 2-3 and 5 stating that there is insufficient antecedent basis for the phrase “the at least one previously disabled portion” in these claims. Specifically, the Final Office Action states that there is only a first portion that is disabled. Appellants respectfully disagree.

Claim 2 recites “selecting at least one previously disabled portion, and automatically re-enabling the at least one selected previously disabled portion.” The “at least one previously disabled portion” recited in claim 2 is not necessarily the “first portion” recited in claim 1, although the first portion could certainly be one of the “at least one previously disabled portion.” The “at least one previously disabled portion” may also be, or at least include, other previously disabled portions that are not the “first portion.” Thus, contrary to the assertion made in the Final Office Action, there is sufficient antecedent basis for the features of claims 2-3 and 5. Accordingly, Appellants respectfully request withdrawal of the rejection of claims 2-3 and 5 under 35 U.S.C. § 112, second paragraph.

In response to this argument, the Examiner in the Advisory Action mailed March 1, 2007 states:

In regards to the rejections of claims 2-3 and 5 under 35 U.S.C. 112, Applicant argues that the “at least one previously disabled portion” may include other previously disabled portions different from the “first portion.” However, there are no other disabled portions recited in the claims other than the “first portion”.

This in no way contradicts Appellants’ argument. Appellants admit that the other “previously disabled portions” are not recited in the independent claim. The argument, however, is that there is no requirement that they be recited in the independent claim.

Claim 2 does not recite “selecting the at least one previously disabled portion” as the Examiner would like to read it. To the contrary, claim 2 recites “selecting at least one previously disabled portion.” Thus, this is a new feature introduced in claim 2 which covers one or more

previously disabled portions. This new feature does overlap the feature of the first portion being disabled and thus, the first portion may be included in “at least one previously disabled portion” or may not be included in the “at least one previously disabled portion.” There is no requirement that every feature of the claims be spelled out in the independent claim, otherwise there would be no purpose in having dependent claims.

All that is required by claim 2 is that at least one previously disabled portion be selected, whether that at least one previously disabled portion includes the first portion or not, and then automatically re-enabling the at least one selected previously disabled portion. There is nothing unclear about the claim. Thus, Appellants respectfully request that the Board overturn the Examiner’s rejection of claims 2-3 and 5 under 35 U.S.C. § 112, second paragraph.

B. Rejection Under 35 U.S.C. § 103(a) Based on Chenier and Storisteneau

The Final Office Action rejects claims 1, 6, 9-14, and 16-21 under 35 U.S.C. § 103(a) as being allegedly unpatentable over Chenier (U.S. Patent Application Publication No. 2004/0003383) in view of Storisteneau (CA 2256931). This rejection is respectfully traversed.

1. Independent Claims 1 and 9-13

Claim 1, which is representative of the other rejected independent claims 9-13 with regard to similarly recited subject matter, reads as follows:

1. A method of editing program code on a data processing system, the program code being suitable for subsequent processing, wherein the method includes the steps of:
 - defining at least two portions of the program code;
 - selecting a first defined portion of the at least two portions of the program code;
 - compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and
 - automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second

defined portion is subjected to subsequent processing.
(emphasis added)

Appellants respectfully submit that neither Chenier nor Storisteneau, either alone or in combination, teach or suggest automatically disabling a first defined portion of program code, the first defined portion being a selected portion whose representation is compressed in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code. Moreover, Appellants respectfully submit that neither Chenier nor Storisteneau, either alone or in combination, teach or suggest making the alleged combination of teachings from Chenier and Storisteneau or the modifications that would be necessary to arrive at the invention as recited in claim 1 or the other rejected independent claims 9-13. Furthermore, Appellants respectfully submit that even if the alleged combination of Chenier and Storisteneau were possible and one were somehow motivated to make the alleged combination, the combination still would not result in the features of the independent claims 1 and 9-13 being taught or suggested.

Chenier is directed to a system for stripping away unwanted portions of program code. Chenier uses a text file to specify information about the types of portions of program code that are to be stripped away and then processes the program code based on the content of this text file. For example, the text file may specify a particular type of tag to look for in the program code such that those portions of code associated with that type of tag are removed from the program code. Chenier does not teach anything regarding a visual representation of the program code, let alone the compression of a representation of a first portion of code in the visual representation of the program code such that the first portion of code is no longer visible in the visual representation of the program code.

The Final Office Action admits that Chenier does not teach compressing a representation of a first defined portion in a visual representation of the program code. However, the Final Office Action alleges that Storisteneau teaches this feature at page 3, lines 15-17 and in Figures 1-2. Appellants respectfully disagree.

Storisteneau teaches a system for providing a graphical representation of code by first providing a graph having nodes representing components of a program and then allowing a user to select nodes to thereby cause the node to be replaced with a window for editing the associated

code portion. Storisteneau is directed to solving the problem of having to have multiple different representations of a program and the user having to correlate information from these multiple different representations.

With the system of Storisteneau, a graph having nodes and arcs, such as shown in Figure 1, is provided. The user may select a node in the graph, and have it replaced with a window for editing the corresponding source code. The user may then edit the source code within the window. Thus, Storisteneau teaches replacing a node representing a portion of code, with a window in which the actual code may be edited. While displaying the window for editing portions of source code may be considered visually representing a portion of program code and returning from the editing window to the nodal representation of the program may be considered a compression of the source code representation into a nodal representation, these are not the same features as are recited in claim 1, as discussed hereafter.

While Storisteneau teaches a graphical representation of code, nowhere in Storisteneau is there any teaching or suggestion of a need to automatically disable a portion of code whose representation in a visual representation of the code has been compressed. Storisteneau is concerned with visualizing the source code for editing while still allowing a user to see the connections between portions of source code via a nodal graph. Storisteneau is not concerned with disabling portions of code, nor does it provide any suggestion to disable portions of code, let alone doing so automatically for portions of code whose representation in a visual representation of a program have been compressed. Moreover, it would not be obvious to modify Storisteneau to include such a feature of automatically disabling portions of code whose representations have been compressed in a visual representation of program code.

The only element in Storisteneau that could reasonably be interpreted as a compressed representation of a portion of code is the node in the graph representation, see Figures 1-2, of Storisteneau for example. If Storisteneau were to automatically disable code whose representation is compressed in a visual representation of the program code, every portion of code that is represented as a node, i.e. in a compressed state, in the graph visual representation would be disabled, i.e. the entire program would be disabled. This would render the system of Storisteneau inoperable and thus, cannot possibly be a teaching or suggestion in the reference.

While Chenier is directed to stripping out portions of code that are associated with tags specified in a text file, Chenier, likewise does not teach or suggest automatically disabling a

portion of code whose representation in a visual representation of program code has been compressed, as has been admitted by the Examiner. Since neither reference alone teaches or suggests this feature, any alleged combination, even if such a combination of the teachings of the references were possible and one were somehow motivated to make the alleged combination, still would not result in the features of the independent claims being taught or suggested.

To the contrary, Appellants respectfully submit that one of ordinary skill in the art would not be motivated to make the alleged combination set forth in the Final Office Action. While both references are directed to editing program code, the teachings of the references are directed to solving completely different problems using completely different solutions and thus, one of ordinary skill in the art would not combine the references in the manner alleged by the Examiner. Chenier is directed to the problem of removing unwanted portions of code and utilizes a text file to identify tags to look for in the code and deleting the portions of code between the tags. Storisteneau is concerned with providing a single representation of code such that portions of code may be edited while the dependencies of code may be visualized through a nodal graph. These are completely different problems from each other, and thus, one of ordinary skill would not even attempt to combine them in the manner alleged by the Examiner.

That is, there is no teaching or suggestion in Chenier regarding any deficiency or desired functionality that the teachings of Storisteneau would satisfy. Similarly, there is no teaching or suggestion in Storisteneau regarding any deficiency or desired functionality that the teachings of Chenier would satisfy. This is because each reference is directed to solving different problems and does not see the need for any mechanism from the other reference in achieving its goal. Thus, there is no teaching or suggestion in the references to make the alleged combination.

The Final Office Action alleges that the motivation is “in order to improve graphical representation of source code and enhance program comprehension” (Storisteneau, page 2, lines 25-27). While this is a nice general goal, it does not address why the specific feature of Storisteneau would be needed or desired in the specific system of Chenier. Where does Chenier state that there is a need to have a nodal representation of program code such that nodes may be selected and their source code displayed? Chenier is not even concerned with the manner by which the program code is represented to a user but instead is concerned with stripping out unwanted portions of code, i.e. code that is not executed and thus, is unnecessary. Moreover, Storisteneau achieves this goal without the need for any of the mechanisms of Chenier and thus,

why would one add the teachings of Chenier to Storisteneau to achieve this purpose?

The only suggestion to even attempt to combine the teachings of Chenier with Storisteneau necessarily comes from a prior knowledge of Appellants' claimed invention and a sole motivation of attempting to recreate the claimed invention having first had benefit of knowing the claimed subject matter. This is impermissible hindsight reconstruction which cannot be used as a proper basis upon which to reject the claims.

Moreover, Appellants respectfully submit that even if the teachings of the references were somehow combinable, and one were somehow motivated to attempt such a combination, *arguendo*, the result still would not be the invention as recited in claim 1 or the other rejected independent claims 9-13. To the contrary, if the teachings of the references were combined, the result would be a system substantially as taught by Chenier where a text file is used to identify tags in code for portions of code that are to be stripped out, but in which the code may be visualized using the graph and edit windows of Storisteneau. There still would be no teaching or suggestion to automatically disable a portion of code that was selected and whose representation in a visual representation of the program code has been compressed. The only teaching regarding such a feature is present in Appellants' claims, not the references.

Thus, for the reasons set forth above, Appellants respectfully submit that neither Chenier nor Storisteneau, either alone or in combination, teach or suggest the features of independent claims 1 and 9-13. Moreover, Appellants respectfully submit that the alleged combination of Chenier and Storisteneau is improper and, even if proper, would not result in the invention as recited in independent claims 1 and 9-13. At least by virtue of their dependency on respective ones of claims 1 and 9-13, the alleged combination of Chenier and Storisteneau does not teach or suggest the features of dependent claims 6, 14, and 16-21. Accordingly, Appellants respectfully request withdrawal of the rejection of claims 1, 6, 9-14, and 16-21 under 35 U.S.C. § 103(a).

2. Dependent Claims 6, 14, and 16-21

In addition to the above, the alleged combination of Chenier and Storisteneau does not teach or suggest the specific features recited in dependent claims 6, 14, and 16-21. For example, with regard to claims 18 and 19, neither reference, either alone or in combination, teaches or suggests that at least one of the at least two portions of the program code has an associated level,

and that selecting a first defined portion of the at least two portions of the program code comprises receiving an input specifying a level such that portions of program code equal to or above the specified level are visually represented in the visual representation of the program code. Moreover, the references do not teach or suggest that portions of the program code that are not equal to or above the specified level are automatically compressed in the visual representation of the program code such that they are not visible.

The Final Office Action alleges that these features are taught by Storisteneau at page 4, lines 1-5, page 7, lines 1-5, and in Figures 2-3. Page 3, line 26 to page 4, line 5 and page 7, lines 1-5 read as follows:

According to a further aspect, the invention provides a method for editing a computer program which consists of the steps of displaying to a user a hierarchical relationship between source code components of the computer program in which each component is represented as a node, and repetitively providing means to allow the user to select one of the nodes for editing and replacing the selected node with an edit window displaying the source code component in order to permit the user to edit the source code component while viewing the hierarchical relationship.

(page 3, line 26 to page 4, line 5)

...example:

- i) a list of other components which call on a component associated with a source node 54 which can be opened for editing from a pop-up menu 55;
- ii) an action that opens a new window (not shown) displaying the topology of the current file or project; and
- iii) an action to refresh the Graph View 61, to reflect changes made to the components of the program.

(page 7, lines 1-5)

These sections of Storisteneau merely describe the hierarchical nature of program code which may be displayed by the system of Storisteneau. Similarly, Figures 2-3 of Storisteneau merely show the hierarchical nodal graph representation or program code. However, nothing in these sections, the figures, or any other portion of Storisteneau teach or suggest selecting a first defined portion of at least two portions of the program code by receiving an input specifying a level such that portions of program code equal to or above the specified level are visually represented in the visual representation of the program code. Moreover, nothing in Storisteneau

teaches or suggests that portions of the program code that are not equal to or above the specified level are automatically compressed in the visual representation of the program code such that they are not visible. Storisteneau only teaches the replacement of nodes in a hierarchical nodal graph with the corresponding source code edit window.

As a further example, regarding claim 20, neither Chenier nor Storisteneau, either alone or in combination, teach or suggest that only portions of code that are visible in a visual representation of program code are stored in a compressed version of program code. The Final Office Action alleges that this feature is taught by Storisteneau at page 18, lines 21-22. However, there is no page 18 in the Storisteneau reference. There is no teaching or suggestion anywhere in Storisteneau to have portions of program code that are visible and portions that are not, but where only the portions that are visible are stored in a compressed version of the program code, as recited in claim 20. Similarly, there is no teaching or suggestion regarding this feature in the Chenier reference. Thus, despite the allegations made by the Examiner in the Final Office Action, there is in fact no teaching or suggestion regarding this feature in any of the cited references.

As another example, with regard to claim 21, neither Chenier nor Storisteneau, either alone or in combination, teach or suggest moving a disabled portion of code from an original position within the program code to a predetermined position within the program code to generate reorganized code and then storing the reorganized program code. The Final Office Action alleges that these features are taught by Chenier at paragraphs 5 and 33 which read as follows:

[0005] The invention provides a process to remove unnecessary information, such as code and/or comments, from one or more files of a computer program. In accordance with one embodiment of the invention, an automated process is provided for removing various types of information from within files of source code. The process obtains an accurate copy of the source code, for example, as it is stored in source control management. The process utilizes one or more text files that define various preprocessor macros and comment flags within the source code. A strip process is then performed, which removes code that is not enabled as well as comments and files that are flagged to be removed, as defined by the text files. The output of the strip process is a source code with the undesired information stripped out. The stripped source code may then be used, for example, to run a build/test suite pass to do validation of the source code before it is ultimately released.

[0033] The Parameters 238 provides various parameter information for the processing software 240 including for example, the input directory where the source code 235 to be stripped is found, the output directory where the stripped source code 255 is to be stored, instructions whether to strip a command or to replace the command with a comment, whether to add a license tag (namely insert a pointer that identifies a root directory for a location of the license file), and/or how the strip process code 245 should be performed depending on whether the purpose of the stripping is for testing or for building code.

All these sections of Chenier teach is the ability to strip code out of the source code and then use the stripped down source code to run a build/test suite. There is nothing in any of these, or any other, sections of Chenier regarding moving code that is disabled within the source code to a predetermined position to generate re-organized code. Thus, despite the allegations raised by the Examiner in the Final Office Action, the references do not in fact teach or suggest the features of claim 21.

The other dependent claims recite additional features which, when taken alone or in combination with the features of their respective independent claims, are not taught or suggested by the alleged combination of references. Thus, Appellants respectfully submit that dependent claims 6, 14, and 16-21 are further defined over the alleged combination of references by virtue of the specific features recited in these claims.

C. Rejection Under 35 U.S.C. § 103(a) Based on Chenier, Storisteneau, and Endejan

The Final Office Action rejects claims 2-5, 8 and 15 under 35 U.S.C. § 103(a) as being allegedly unpatentable over Chenier in view of Storisteneau, and further in view of Endejan (U.S. Patent Application Publication No. 2002/0184611). This rejection is respectfully traversed.

1. Independent Claims 1 and 9-13

Chenier and Storisteneau suffer from the deficiencies discussed above, i.e. neither Chenier nor Storisteneau, either alone or in combination, teach or suggest automatically disabling a portion of code whose representation has been compressed in a visual representation of the

program code, as recited in the independent claims. Endejan, likewise, does not teach or suggest such a feature.

Endejan teaches a system in which active and inactive portions of program code are displayed using different fonts and grayscaling. With Endejan, pre-processor directives are used to specify which portions of the program code are active and which are inactive. Based on these pre-processor directives, the active code is displayed in one font and grayscale while the inactive code is displayed in a different font and/or grayscale. The inactive and active code portions may be switched by changing the pre-processor directives (see paragraphs 25-30).

Endejan, like Chenier and Storisteneau, does not teach or even suggest automatically disabling a portion of code whose representation has been compressed in a visual representation of the program code. With the Endejan system, inactive portions of code are displayed with a different grayscale or font. Endejan does not compress the representation of portions of code and then disable the portions of code whose representation has been compressed. To the contrary, Endejan must know ahead of time which portions of code are inactive so that it may properly represent them as having a different font and/or grayscale. Thus, none of the cited references, Chenier, Storisteneau, or Endejan, either alone or in combination, teach or suggest the specific features of claim 1, from which claims 2-5, 8 and 15 depend. Therefore, claims 2-5, 8 and 15 are likewise allowable over the asserted combination of Chenier, Storisteneau, and Endejan for at least the reasons set forth above with regard to claim 1. Accordingly, Appellants respectfully request withdrawal of the rejection of claims 2-5, 8 and 15 under 35 U.S.C. § 103(a).

2. Dependent Claims 2-5, 8 and 15

In addition to the above, neither Chenier, Storisteneau, nor Endejan, either alone or in combination, teach or suggest the specific features of dependent claims 2-5, 8 and 15. For example, with regard to claim 3, the references do not teach or suggest to assign each defined portion of program code to a category of a set including at least one category, the step of selecting the first defined portion and the step of selecting the at least one previously disabled portion including selecting at least one category. The Final Office Action alleges that this feature is taught by Chenier at paragraph 5. Paragraph 5 of Chenier merely discusses the use of a text file to define preprocessor macros and comment flags within the source code that identifies

portions of source code to be removed through a stripping process. Chenier does not teach or suggest that each portion of program code is assigned a category and that these categories are used to select a portion of code whose representation in a visual representation of the program code is to be compressed and the corresponding portion of code disabled. Neither does paragraph 5 of Chenier teach or suggest that these categories may be used to select at least one previously disabled portion that is to be re-enabled.

Similarly, with regard to claim 4, neither Chenier, Storisteneau, nor Endejan, either alone or in combination, teach or suggest that the set of at least one category includes a category for service instructions. The Final Office Action alleges that this feature is taught by Chenier because Chenier mentions testing and debugging in paragraph 3. This section of Chenier merely mentions that developers may include code in a program for testing or debugging purposes and that once the testing or debugging is done, the code is no longer necessary. However, nowhere in Chenier is there any teaching or suggestion to provide a set of at least one categories for code with one of the categories being for service instructions. Thus, despite the allegations raised by the Final Office Action, the cited references in fact do not teach or suggest the features of claim 4.

As a further example, with regard to claim 15, neither Chenier, Storisteneau, nor Endejan, either alone or in combination, teach or suggest re-enabling a first defined portion by removing comment tags associated with service instructions. The Final Office Action alleges that this feature is taught in paragraph 29 of Endejan which reads:

According to yet another advantageous feature, design system 10 is configured to automatically switch the display format of active code segment 36 to an inactive display format, as shown by the display formats of inactive code segments 40 and 42, in response to receiving a change to the set of pre-processor directives 34. Further, editor 14 is configured to automatically switch the display format of one or more of inactive code segments 40 and 42 from the inactive display format to the active display format in response to a change to the set of pre-processor directives 34. This change may happen in real time and without the need for an additional compilation step or other operator input. Pre-processor directives 34 need not be in the same source code file for this feature to work, but could instead be located in one or more other files included by reference, as is indicated by the pre-processor directive “#include”.

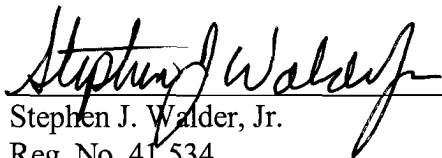
Nothing in this section of Endejan teaches or even suggests to re-enable a disabled portion by removing comment tags, as recited in claim 15. All this section of Endejan teaches is that the display format of active and inactive code segments may be switched in response to changes to pre-processor directives. There is nothing in Endejan regarding comment tags or the ability to re-enable a previously disabled portion of code by removing the comment tags as recited in claim 15.

The other dependent claims recite additional features which, when taken alone or in combination with the features of their respective independent claims, are not taught or suggested by the alleged combination of references. Thus, Appellants respectfully submit that dependent claims 2-5, 8 and 15 are further defined over the alleged combination of references by virtue of the specific features recited in these claims.

VIII. Conclusion

In view of the above, Appellants respectfully submit that the features of claims 1-6 and 8-21 are not taught or suggested by the alleged combination of references. Accordingly, Appellants request that the Board of Patent Appeals and Interferences overturn the rejections set forth in the Final Office Action.

Respectfully submitted,


Stephen J. Walder, Jr.
Reg. No. 41,534
Walder Intellectual Property Law, P.C.
P.O. Box 832745
Richardson, TX 75083
(214) 722-6419
ATTORNEY FOR APPELLANTS

CLAIMS APPENDIX

1. A method of editing program code on a data processing system, the program code being suitable for subsequent processing, wherein the method includes the steps of:
 - defining at least two portions of the program code;
 - selecting a first defined portion of the at least two portions of the program code;
 - compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and
 - automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.
2. The method according to claim 1, further including the steps of:
 - selecting at least one previously disabled portion, and
 - automatically re-enabling the at least one selected previously disabled portion.
3. The method according to claim 2, further including the step of:
 - assigning each defined portion to a category of a set including at least one category,
 - the step of selecting the first defined portion and the step of selecting the at least one previously disabled portion including selecting at least one category.

4. The method according to claim 3, wherein the set includes at least one category for service instructions.
5. The method according to claim 2, wherein the program code includes a plurality of instructions, the step of automatically disabling the first selected portion including converting each corresponding instruction into a comment, and the step of automatically re-enabling the at least one selected previously disabled portion including restoring each corresponding instruction.
6. The method according to claim 1, wherein the step of defining the at least two portions of the program code includes:
 - enclosing each portion between a starting comment and an ending comment.
8. The method according to claim 1, further including the steps of:
 - updating the program code by removing the first defined portion, and
 - storing the updated program code.
9. A computer readable medium, having a computer readable program encoded thereon and being directly loadable into a working memory of a data processing system, the computer readable program having instructions for performing a method of editing program code when the computer readable program is run on the data processing system, the program code being suitable for subsequent processing, wherein the method includes the steps of:
 - defining at least two portions of the program code;
 - selecting a first defined portion of the at least two portions of the program code;

compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and

automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.

10. A program product comprising a computer readable medium on which a computer program is stored, the program being directly loadable into a working memory of a data processing system for performing a method of editing program code when the program is run on the data processing system, the program code being suitable for subsequent processing, wherein the method includes the steps of:

defining at least two portions of the program code;

selecting a first defined portion of the at least two portions of the program code;

compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and

automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.

11. An editor for editing program code on a data processing system, the program code being suitable for subsequent processing, wherein the editor is provided as a computer readable program on a computer readable medium, wherein the computer readable program includes software instructions for:

defining at least two portions of the program code;

selecting a first defined portion of the at least two portions of the program code;

compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and

automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.

12. A data processing system for editing program code, the program code being suitable for subsequent processing, wherein the system includes:

means for defining at least two portions of the program code;

means for selecting a first defined portion of the at least two portions of the program code;

means for compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program

code; and

means for automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.

13. A data processing system for editing program code, the program code being suitable for subsequent processing, wherein the system includes:

a processor;

a memory coupled to the processor; and

an input device, wherein the memory comprises instructions which, when executed by the processor, cause the processor implement:

a software module for defining at least two portions of the program code;

a software module for selecting a first defined portion of the at least two portions of the program code in response to an input from the input device;

a software module for compressing a representation of the first defined portion in a visual representation of the program code such that content of the first defined portion is not visible in the visual representation of the program code, wherein a second defined portion of the at least two portions of the program code remains visible in the visual representation of the program code; and

a software module for automatically disabling the first defined portion, the disabled first defined portion being excluded from the subsequent processing, wherein the second defined portion is subjected to subsequent processing.

14. The method of claim 1, wherein the first defined portion is a service instruction portion, and wherein disabling the service instruction portion comprises automatically converting the service instructions in the service instruction portion to comments in the program code by inserting comment tags in association with the service instructions.

15. The method of claim 14, further comprising:
receiving an input to re-enable the first defined portion; and
automatically re-enabling the first defined portion in response to receiving the input, wherein re-enabling the first defined portion comprises removing the comment tags associated with the service instructions.

16. The method of claim 1, wherein compressing the representation of the first defined portion in the visual representation of the program code comprises:

replacing a visual representation of the content of the first defined portion with an identifier of the first defined portion, the identifier indicating a position in the program code where the first defined portion was present but not containing contents of the first defined portion; and

inserting, into the visual representation of the program code, a compression identifier in association with the identifier of the first defined portion, the compression identifier indicating that the first defined portion has been compressed.

17. The method of claim 16, wherein the compression identifier is user selectable, and wherein, in response to a user input selecting the compression identifier, the contents of the first

defined portion are expanded in the visual representation of the program code and are re-enabled.

18. The method of claim 1, wherein at least one of the at least two portions of the program code has an associated level, and wherein selecting a first defined portion of the at least two portions of the program code comprises receiving an input specifying a level such that portions of program code equal to or above the specified level are visually represented in the visual representation of the program code, and wherein portions of the program code that are not equal to or above the specified level are automatically compressed in the visual representation of the program code such that they are not visible.

19. The method of claim 18, wherein the first defined portion and second defined portion are both of a same content type but have different associated levels.

20. The method of claim 1, wherein only portions of the program code that are visible in the visual representation of the program code are stored in a compressed version of the program code.

21. The method of claim 1, wherein the first defined portion of the program code is a comment in the program code, the method further comprising:

moving the first defined portion from its original position in the program code to a predetermined position within the program code to thereby generate re-organized program code;
and

storing the re-organized program code.

EVIDENCE APPENDIX

NONE

RELATED PROCEEDINGS APPENDIX

NONE